DOCUMENT RESUME

ED 174 466 SE 028 538

AUTHOR	Weissman, Kenneth
TITLE	School BASIC.
INSTITUTION	Dartmouth Coll., Hanover, N.H. Kiewit Computation Center.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	Feb 70
GRANT	NSF-GW-2246
NOTE	87p.; For related documents, see SE 028 535-537; Not available in hard copy due to copyright restrictions; Contains occasional light and broken type
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Computer Oriented Programs; *Computer Programs; *Computers; Curriculum; Instruction; *Learning Activities; *Mathematics Education; Problem Sets; *Programing; Secondary Education

ABSTRACT

This booklet was designed for the use of junior high school or high school students. Detailed, step-by-step instructions are given for using the computer, for writing programs in BASIC, and for using the various commands in the BASIC language. A list of 75 computer programing ideas in varying degrees of difficulty is presented in the index. (MP)

School BASIC

by

Kenneth Weissman
Benjamin Franklin High School

SECONDARY SCHOOL PUBLICATION

KIEWIT COMPUTATION CENTER
DARTMOUTH COLLEGE
HANOVER, NEW HAMPSHIRE 03755

(Supported in part by National Science Foundation Grant No. GW-2246)

February, 1970

TABLE OF CONTENTS

FOREWARD TO TEACHERS

This booklet was designed for the use of Junior High School and High School students. A student who shows an interest in computers or has completed more than a semester of High School Algebra should be given the opportunity to use the regular BASIC Manual (latest edition).

While a limited number of exercises and program examples are given in this booklet, sets of TOPIC OUTLINES are available from:

NSF SECONDARY SCHOOL PROJECT
Kiewit Computation Center
Dartmouth College
Hanover, New Hampshire 03755

NOT ALL OF THE BASIC LANGUAGE IS EXPLAINED IN THIS BOOKLET.

Explanations of BASIC and other mathematical concepts contained herein have been adjusted to meet the needs and understanding of the "average" teenager.

A table of contents and index is provided for easier reference.

The specific individual ability, imagination, and creativity of the student determines to a large extent the meaningful and productive use of a time-sharing teletype in school. A high degree of teacher interest also leads to motivated students.

Ken Weissman

- 0 -

Preface

"A program is a set of directions that is used to tell a a computer how to provide an answer to some problem."

This booklet will explain many of these directions, so that you will be able to operate the computer. The directions are called a language.

The language that this booklet explains is called BASIC. It is a lot of fun to learn. However, not everything about the BASIC language is explained in this booklet. When you finish the booklet you might like to learn more. If you do, then ask your teacher for more information.

The computer you are using is a GE-635 located in the Kiewit Computation Center at Dartmouth College, Hanover, New Hampshire. You will be able to operate the Dartmouth Time-Sharing System (DTSS) computer from a teletype terminal at your school that uses regular telephone lines for communication.

CHAPTER ONE

USER NUMBER

Everybody needs a user number.

Get your user number from your teacher, or the person in charge. Write it below so you won't forget.

My user number:_____ _____

TURNING ON THE TELETYPE

Directions are posted near most teletypes telling how to operate them.

There are several different models of teletype machines.

Usually you just push the originate button, marked ORIG. Some teletypes are marked LINE 1 or LINE 2, instead of ORIG. In a few seconds, you will hear a beep, and you are ready to start.

On some hook-ups, you may hear a dial-tone after pushing the ORIG button. If this happens, then dial anyone of these special phone numbers (don't pick up the phone):

7 6511 2091 2101

Then in a few seconds, you will hear a beep, and you are ready to start.

If you have any trouble, please let your teacher know about it, so that you can be helped.

NEW

After turning on the teletype, hearing the beep, and giving your user number, the computer will ask NEW OR OLD--.

If you are starting a NEW program, type NEW, and push the return key on the teletype.

(You must push the return key every time you have finished typing a line.)

The computer will then ask for the NEW FILE NAME--.

Make up a name. (You can name your program with any word up to 8 letters.)

Given below are some sample program names.

JOE FAT-2 X JACK BASEBALL

NOW YOU'RE READY TO WRITE A PROGRAM IN BASIC

The computer isn't very smart. You have to tell it everything it has to do, in a specific order.

The line numbers tell the computer in which order to do the program, usually lowest number first. Line numbers also make it easy for us to locate parts of the program so that we can make changes.

If you didn't already know that the computer could:

Add	(+)	
Multiply	(*)	
Divide	(/)	
Subtract	(-)	then you have just been informed.

Let's write a program that will multiply 15 by 3.

O.K.!! Here goes the program!

```
DARTMOUTH TIME-SHARING
TERMINAL 124 ON AT 10:49  12 AUG 69, 058 USERS
DTSS TILL 2400.  LIST CCNEWS*** 6 AUG 69
```

} Heading typed by teletype.

```
USER NUMBER--(Type your user number)
NEW OR OLD--NEW TIMES
READY
```

(remember to push the return key at the end of every line)

```
10 PRINT 15*3
20 END
RUN

TIMES    08/12/69  10:50

 45

TIME: 0.038 SEC.
READY
```

That's all there is to it! After a second or so, the teletype will print your answer....45.

With this little bit of information, you can probably do most of the arithmetic you will ever need!

It's a simple job to change line 10 to either add (+), subtract (-), or divide (/). It is just as easy to change the numbers you want to work with.

Here is another program that adds three numbers 8,3, and 7:

```
NEW ADD
READY
                        (remember you must push the return key at
                        the end of each line.)
10 PRINT 8+3+7
20 END
RUN

ADD      08/12/69  10:51

 18

TIME:  0.040 SEC.
READY
```

LINE NUMBERS

All programs have line numbers.

The line number identifies the line and tells the computer which lines to do in order. (Usually, lowest number first.)

We try to leave space between line numbers, so that we can place other lines between. (10,20,30 instead of 11,12,13, etc.)

A set of line numbers is shown below:

```
10
20
30      you can choose any set of line numbers you want.
35
59
80
etc.
```

When you have finished typing a line, you must push the return key on the teletype keyboard.

Line numbers may be typed in any order.

```
 8                                                    8
32  the computer will sort them from lowest to      15
15  highest---------------------------------->      19
19                                                   32
```

You may eliminate any line simply by retyping the line number with nothing after it. (This is useful when making corrections.)

You may choose any set of line numbers you wish. Most people pick line numbers in the ten times table. (10,20,30,40,etc.)

MAKING CORRECTIONS

There are 5 major ways to make corrections when you are working on the teletype that is hooked into the DTSS (Dartmouth Time-Sharing System):

1. To eliminate the line you are working on, you can just retype the line number and start all over again.

2. To delete the line that you are working on, push the control key and the X key at the same time.

3. To delete one or two, or just a few letters, push the shift keyand the O key at the same time. This produces a backwords arrow ← for each letter eliminated. An example is shown below:

SATRUDAY←←←←←URDAY

The computer will continue as if RUDAY was never typed.

4. You can type NEW at any time and eliminate or erase your entire current program.

5. You can type IGNORE at any time and everything since your last command will be ignored. Some commands are: RUN, LIST, SAVE, UNSAVE, REPLACE, NEW, OLD.

END

The last line of a program must contain the END statement. It looks like this:

999 END

Only one END is allowed in a program.

(The computer isn't too smart, and you have to tell it where the end of the program is.)

PRINT - Calculations

Calculations involving addition (+), subtraction (-), multiplication (*), or division (/) can be done with a PRINT statement.

Other calculations which will be explained later can also be done using the PRINT statement in the same manner. The names of some of these are:

Exponents (↑), Square root SQR(x), Sine SIN(x), Cosine COS(x), Tangent TAN(x), absolute value ABS(x), exponents to base e EXP(x), and complicated formulas that you can make up.

In the example below, the number 10 is divided by 2.

```
NEW JOE
READY
            (remember you must always push the return key at the
            end of every line.)
15 PRINT 10/2
20 END
RUN

JOE         08/12/69  10:53

 5

TIME:  0.039 SEC.
READY
```

The program will print the answer...5 after the RUN command is typed. The calculation of 10/2 is done by the computer.

Can you write a program that adds two numbers together?

Can you write a program that multiplies three numbers together?

PARENTHESES

The computer isn't able to determine what you really wanted to do if you make a mistake. It can only do what you tell it to do.

Suppose you wanted to find the average of two test grades 70 and 90. If you wrote this program like someone I know did, you would get the wrong answer.

WRONG

```
NEW AVERAGEW
READY

10 PRINT 70+90/2
20 END
RUN

AVERAGEW  08/12/69  10:54

 115

TIME:  0.041 SEC.
READY
```

The computer printed out 115 and we know this is wrong.

What we really wanted to do was to add 70+90 first, and then divide by 2. This can be done by using parentheses.

CORRECT

```
NEW AVERAGEC
READY

10 PRINT (70+90)/2
20 END
RUN

AVERAGEC  08/12/69  10:54

 80

TIME:  0.041 SEC.
READY
```

The computer will print out 80, the correct answer.

What happened here was that the computer first added 70+90 and got 160, then divided the 160 by 2 to get 80.

The computer is scheduled to do multiplication and division before addition and subtraction and then to proceed from left → to right

in calculating answers.

Suppose we wanted to add 2+2 and then multiply the result by 3. If we simply wrote: 2+2*3 or 3*2+2, we would get the same wrong answer in both cases:

```
NEW TRY1
READY

10 PRINT 2+2*3
20 END
RUN

TRY1   08/12/69   10:55

 8

TIME:  0.041 SEC.
READY
```

or

```
NEW TRY2
READY

10 PRINT 3*2+2
20 END
RUN

TRY2   08/12/69   10:55

 8

TIME:  0.041 SEC.
READY
```

Of course, this isn't what we wanted to do. To correct this, we use parentheses:

```
NEW TRY
READY

10 PRINT (2+2)*3
20 END
RUN

TRY   08/12/69 10:56

 12

TIME:  0.039 SEC.
READY
```

A good rule to follow is that if you aren't sure what the computer will do, group your calculations with parentheses the way you want the problem solved.

A more complicated problem is shown below:

Suppose you wanted to add 3+5 and 2+7, then take both of these answers and multiply them together, after doing that you want to divide by the number 4. What expression would do that?

PARENTHESES (Continued)

```
NEW PARENTHE
READY

10 PRINT ((3+5)*(2+7))/4
20 END
RUN

PARENTHE  08/12/69  10:57

 18

TIME:  0.040 SEC.
READY
```

The answer that the computer will calculate is....18.

PRINT - Numbers

Suppose you just want to print a number. The PRINT statement can be used for this purpose.

```
NEW NUMBER
READY

10 PRINT8
20 END
RUN

NUMBER  08/12/69  10:58

 8

TIME:  0.039 SEC.
READY
```

The program will print the number 8 after you type the word RUN.

Write a program that will print the number 143.

```
NEW NUMBER1
READY

10 PRINT 143
20 END
RUN

NUMBER1  08/12/69  10:58

 143

TIME:  0.040 SEC.
READY
```

When the computer asks NEW OR OLD and you are calling-up a program that you have already worked on and saved, type the word OLD, and push the return key.

The computer will then ask for the OLD FILE NAME--.

Type the old program name exactly as you did the first time you used it.

Some OLD programs that have already been worked on and saved are given below (try them):

BANDIT*** BINGO*** (***indicates placed in the computer library)

CHANGING THE PROGRAM YOU ARE WORKING ON

At any time you may type NEW or OLD. This will allow you to start a new program or call-up an old program from a computer library called catalog.

Once you have a little experience, you can take a shortcut and call a program directly.

Two examples of this are shown below:

OLD BANDIT*** (return key)

The computer will answer READY or

NEW JOE (return key)

The computer will answer READY.

S KEY (STOP)

At any time, even when the teletype is printing, you may stop the program simply by pushing the letter S key on the keyboard, or typing the word STOP.

Certain other commands you will find useful are given below:

'RUN Command'

When you type RUN, your program will do the job (be executed), if properly written.

If not written correctly, the computer will tell you some of the errors, and where to look for them.

'LIST Command'

When you type LIST, your program (as written and corrected) will be listed on the teletype.

'SAVE Command'

If you wish to SAVE your program for some later time, type the word SAVE. The program is placed in an area in the computer called storage, or library, or catalog.

'UNSAVE Command'

If you wish to remove your program from storage (library or catalog) and UNSAVE it, type the word UNSAVE. This is almost the same as DESTRUCT in the TV program "Mission Impossible."

'REPLACE Command'

If you have already saved a program and wish to correct or modify it, type the word REPLACE or REP after making the latest correction.

'HELLO Command'

By typing HELLO or HEL you can change the user number without turning off (disconnecting) the telephone line.

'BYE or GOODBYE Command'

When typing BYE or GOODBYE, this is a signal to the computer that you wish to discontinue your work. The computer will then disconnect the teletype and erase everything you have done. You can avoid erasure by typing SAVE OR REP before typing BYE.

Now that you are an Official BASIC Programmer, Level One, you are ready to start on Level Two.

If you tried either:

OLD BANDIT*** or

OLD BINGO*** described earlier, you realize that there is more to programming than just adding or multiplying numbers together.

First, these programs used words and whole sentences, while you have used only numbers.

Second, these programs skipped or jumped lines and asked you questions.

This chapter will explain how some of this was done, so you can do it too.

A message to be printed must have quotes (" ") around it. Also, everything inside the quote symbols will be printed as typed.

A program to print the message SNOOPY SLEEPS is given below:

```
NEW SLEEPS
READY

30 PRINT "SNOOPY SLEEPS"
40 END
RUN

SLEEPS  08/12/69  10:59

SNOOPY SLEEPS

TIME:  0.040 SEC.
READY
```

Notice the quote symbols before and after the message

Write a program that will print your name.

Write a program that will print the name of your school.

You can have more than one PRINT statement in a program.

Shown below are two programs that print a message about Snoopy. The first program has a semi-colon, the second program doesn't.

The semi-colon (;) at the end of line 32 causes the next line to print right after it. Try the program both ways, with and without the semi-colon (;).

```
NEW SNOOP-1
READY

32 PRINT "SNOOPY SLEEPS";
34 PRINT "UNDER THE TREE."
39 END
RUN

SNOOP-1  08/12/69  11:02

SNOOPY SLEEPS UNDER THE TREE.

TIME:  0.049 SEC.
READY
```

```
NEW SNOOP-2
READY

32 PRINT "SNOOPY SLEEPS"
34 PRINT "UNDER THE TREE."
39 END
RUN

SNOOP-2  08/12/69  11:03

SNOOPY SLEEPS
UNDER THE TREE.

TIME:  0.049 SEC.
READY
```

MORE ON PRINT (Continued)

It is very important to realize that the semi-colon(;) causes information to be squeezed or placed on the same line.

The comma (,) not shown on the preceding page places information fifteen spaces apart in 5 columns. The comma will be explained in greater detail later.

LINE JUMPING USING PRINT

We just learned that the PRINT statement permits the computer, by means of the teletype, to print or write a message.

We can use the PRINT statement in other ways.

The PRINT statement with nothing after it (without a message) is an order to make the paper on the teletype move-up one line.

An example of this program is given below:

```
20 PRINT                    Paper moves up one line
30 END                      on the command RUN.
```

Can you write a program to jump the paper five lines?

Don't look at the answer!!! Try it first.

Most probably you chose the following program or something like it to make the paper jump five lines:

```
NEW JUMP-1
READY

21 PRINT
22 PRINT
23 PRINT
24 PRINT
25 PRINT
30 END
```

However, an easier method is available using the line-feed key.

Here it is:

```
21 PRINT "
                    Notice the line-feed key was used
                    between the quote symbols.
         "          Try it.
30 END
```

WHAT IS A VARIABLE?

A variable in the BASIC language is any one of the 26 letters of the alphabet.

Each letter (or variable) may be given a different number value at various points in our program.

However, we try to avoid using the letter 'O' (oh), because we can easily mix this letter up with the number zero.

Some examples of variables are given below:

A B C R S T W X Y Z

Can you name any three other variables not already shown?

SOME ADDITIONAL VARIABLES

If we use up all of the letters in the alphabet, the BASIC language has additional variables we can use.

Every letter followed by a number from 0 to 9 is also considered a variable. This combination of letter and number is treated just as if it were a single letter.

Some examples are given below:

A1 A6 H9 I7 Z5 B1 K3 L8

Is K25 a variable?	NO, because the letter is followed by a 2 digit number.
Is N6 a variable?	YES.
Is O3 a variable?	YES, but we try to avoid using the letter 'O' (oh), because we easily mix it up with the number zero.

THE LET STATEMENT USES VARIABLES (or letters)

So far, we have not assigned any number value to a letter.

The LET statement permits you to assign a value (some number) to a letter (variable).

A program using the LET statement is shown below:

```
NEW LET1
READY

10 LET X=5                          ' X IS ASSIGNED THE VALUE OF 5.
20 PRINT X                          ' THE 5 IS PRINTED.
30 END                              ' THE PROGRAM STOPS.
RUN

LET1  08/12/69  11:08

 5

TIME:  0.041 SEC.
READY
```

We didn't put a quote (" ") around X because we want the value of the variable X printed, not the letter X. Another way of saying this is: we want the number value assigned to X printed; in this case it was 5.

Here is another program:

```
NEW LET2
READY

10 LET X=5                           ' X IS ASSIGNED THE VALUE 5.
20 PRINT "X="X                       ' THE 5 IS PRINTED, SO IS X=.
30 END                               ' THE PROGRAM STOPS.
RUN

LET2  08/12/69  11:09

X= 5

TIME:  0.045 SEC.
READY
```

Here the program will print out: X=5

Another program using the LET statement and a different variable is shown below:

```
NEW LET3
READY

10 LET R3=5                    ' R3 IS ASSIGNED THE VALUE 5.
20 PRINT R3                    ' THE 5 IS PRINTED.
30 END                         ' THE PROGRAM STOPS.
RUN

LET3  08/12/69  11:10

 5

TIME:  0.044 SEC.
READY
```

Given below is a more complicated program that multiplies TWO letters (variables) together:

```
NEW LET4
READY

NEW LET4
READY

10 LET A=15
20 LET B=3
30 LET C=A*B
40 PRINT C
50 END
RUN

LET4  08/12/69  11:11

 45

TIME:  0.042 SEC.
READY
```

or

```
NEW LET5
READY

10 LET A=15
20 LET B=3
30 PRINT A*B
40 END
RUN

LET5  08/12/69  11:12

 45

TIME:  0.044 SEC.
READY
```

The answer 45 is printed by the teletype.

STRINGS ($)

We can let a variable be equal to an entire word or sentence, just by putting a dollar sign ($) after the letter.

This is an example of a string:

```
NEW STRING
READY

10 LET A$="SNOOPY SLEEPS"
20 LET B$="UNDER THE TREE."
30 PRINT A$;B$
40 END
RUN

STRING  08/12/69  11:13

SNOOPY SLEEPSUNDER THE TREE.

TIME:  0.054 SEC.
READY
```

Remember the semi-colon keeps the message on the same line. Close-up or tightly packed on the line.

The teletype will print:

SNOOPY SLEEPSUNDER THE TREE

We goofed! We didn't leave a space after SLEEPS, so the lines came together.

Let's fix it up. Just type line 10 correctly.

```
10 LET A$="SNOOPY SLEEPS "
RUN

STRING  08/12/69  11:13

SNOOPY SLEEPS UNDER THE TREE.

TIME:  0.052 SEC.
READY
```

Now the teletype printed:

SNOOPY SLEEPS UNDER THE TREE

I'm certain that you will want to try out strings for a little while before we go on to the next part of being a Computer Programmer Level Two.

Try printing your name using a string, or perhaps write a whole paragraph.

ASKING QUESTIONS (INPUT STATEMENT)

We can make a program ask a question, perhaps even ask your name and age. Below is a sample program that does just this:

```
NEW RED
10 PRINT "HI, MY NAME IS GE-635"
20 PRINT "WHAT IS YOUR NAME";
30 INPUT A$
40 PRINT "HOW OLD ARE YOU"
50 INPUT A
60 END
RUN
```

The computer will print:

```
HI, MY NAME IS GE-635
WHAT IS YOUR NAME?          and stop.
```

Notice the semi-colon causes the question mark (?) to be placed right after WHAT IS YOUR NAME. But, no question mark was in the program!!! Where did the question mark come from?

The INPUT statement causes a question mark (?) to be printed.

Notice we used a variable A$, because we expect to receive a word message (or string) back after the computer stops. The computer stops after the ? is printed on the teletype paper.

You answer: (Your own name or RED BARON, or whatever)

```
RED BARON (return key)
HOW OLD ARE YOU
?                   and stop.
```

Notice the semi-colon is missing on line 40, so the INPUT question mark (?) appears on the next line, and the teletype stops. Also, the variable this time was A, because we expect a number and not a word message.

You answer: (Your age, or 99, or some number) and (return key)

Can you write a program using INPUT statements?

Given below are two more useful statements that we will use shortly.

STOP

The STOP statement acts like the END statement, by ending or stopping the program.

STOP statements can be located anywhere, except the last line.

Examples of a STOP statement look like this:

```
10
20 STOP
30
37 STOP
40
45
90
110 END
```

The program has 2 STOP statements.

The END statement is always the last line.

GO TO

The GO TO statement permits the computer to jump around in a program and not follow the order of line numbers from lowest to highest.

An example of this is given below:

```
30 GOTO 47
41 STOP
47
53 GO TO 30
99 END
```

When the computer reaches line 30, it is told to go to line 47, jumping over the STOP on line 41.

At line 53, it is told to go back to line 30, starting the process all over again.

This is called loop.

Will this program ever end?

How can you break (get out of) the loop?

CONGRATULATIONS!!!!!!!!!!!!!!!!

You are now an Official BASIC Programmer, Level Two. Of course, to be a Level Three Programmer requires more work on your part, and a lot more math.

This chapter deals with something called exponents, which is nothing more than a shorthand way of multiplying the same number together.

Also, this chapter works with fractions and explains what happens to them.

I'm certain that you would want to know more about loops which make things easier to program, and how to display your answers in a neater fashion.

EXPONENTS

In arithmetic, we have a shorthand way of saying 4 times 4. It is written 4^2. The little ^{2} is called an exponent. Of course, the answer is 16.

Also,

7X7	is	7^2	or 49
8X8	is	8^2	or 64
10X10X10	is	10^3	or 1000
2X2X2X2X2	is	2^5	or 32

What is 3^2? 5^2? 4^3?

In the BASIC language, we use the upward arrow (↑) to denote the exponent.

Suppose we wanted to find the area of a square whose side was 6, we could then write a program to do this:

```
NEW SQUARE
READY

10 PRINT 6↑2
20 END
RUN

SQUARE  08/12/69  11:16

 36

TIME:  0.040 SEC.
READY
```

Since the area of a square is its length times width, and in this case both were six, we could have written 10 PRINT 6*6 and have gotten the same answer.

SIGNS (+ and -)

Before every number in the BASIC language is a space for the sign. If it's positive, the sign is NOT printed and the space is left blank or empty. However, if it is negative or minus (-), the sign is put in by the computer.

All numbers must be either positive (+) or negative (-) in BASIC. Zero, of course, doesn't have a sign, but it does have the space.

FRACTIONS AND SCIENTIFIC NOTATION

Fractions

The computer understands the symbol (/) meaning division, as a fraction. However, all answers to problems are given as either:

(a)	Integers	0,1,2,3,-5,etc.
(b)	Decimals	0.1, .325, -.4,etc. or as an
(c)	Exponent	1.4E+9 (meaning 1400000000) or 1.2E-3 (meaning .0012)

Scientific Notation

Numbers larger than 8 digits are converted into the E notation, as well as decimals smaller than one-tenth (0.1). Some examples of this are given below:

(a) The number 45,176,325,416 has 11 digits. The computer will then change the number to: 4.51763E+10

(b) The number 39,165,216 will be printed as: 39165216

(c) The number 0.34561273215 will be printed as: 3.45612E-1

From the above, it can easily be shown that E+2 means 'times 100' or 10^2 and E+3 means 'times 1000' or 10^3. E-2 means 'divide by 100' or divide by 10^2.

This topic is usually called SCIENTIFIC NOTATION and additional questions about it should be asked of your teacher. It was presented here so that in the event you did get a number with E, you would be somewhat familiar with this strange type of answer and could find out more about it at that time.

FOR...NEXT LOOP

Loops make counting easier, as well as some other types of problems.

Suppose we wanted to print a series of numbers 1,2,3,4,5, etc. to 100. (Usually, this is written as 1,2,3,.. 100).

We could type out each number after a print statement something like this:

```
5 PRINT 1
10 PRINT 2
15 PRINT 3
etc.
```

This is very time consuming, but it would work!

There is a shorter way of doing this using a loop. In this case, the loop is called a FOR...NEXT loop because we use the words FOR and NEXT.

For every FOR there must be a NEXT in the program.

Here's how to use the FOR...NEXT statements:

Choose any letter (a dummy variable). I like the letter J. You can choose any letter you want.

```
NEW LOOP
READY

10 FOR J=1 TO 100
20 PRINT J;
30 NEXT J
40 END
RUN

LOOP  08/12/69  11:17

 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
 39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
 57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
 75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92
 93  94  95  96  97  98  99  100

TIME:  0.257 SEC.
READY
```

I put the semi-colom (;) after J in line 20 because I wanted the numbers close-packed. A comma (,) would put 5 numbers on a line. Nothing after J would put the numbers underneath each other.

What happens here is that the J takes the value of 1, is printed by line 20, and sent back to line 10. The word NEXT acts like 'GOTO line 10'.

However, we already used 1, so it takes the next number 2, prints 2 and is sent back to line 10 by the NEXT. This continues, round and round (loop), until all the J's are used up.

When all the J's are used up, and we come to the NEXT, there is no NEXT J so the program goes onto line 40 which in this case is the end.

Suppose I wanted to write a program that would print out all of the whole numbers between 8 and 17. It's a simple matter using a FOR...NEXT loop to do this!

```
NEW LOOP-1
READY

10 FOR J=8 TO 17
20 PRINT J,
30 NEXT J
40 END
RUN

LOOP-1   08/12/69   11:19

 8              9              10             11             12
 13             14             15             16             17

TIME:  0.090 SEC.
READY
```

In this case, the numbers will be printed 5 to a line, 15 spaces apart because a comma (,) after J on line 20 was used.

Shown below are a number of other examples that can be done simply by retyping line 10.

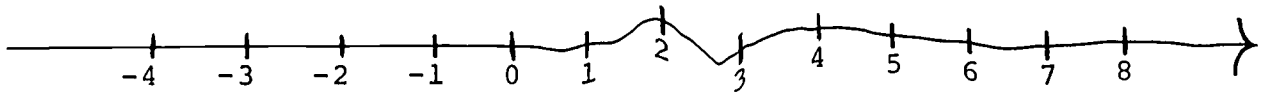
(a) 10 FOR J=0 TO 4
(b) 10 FOR J=-5 TO 5
(c) 10 FOR J=-10 TO 0
(d) 10 FOR J=-100 TO 0
(e) 10 FOR J=100 TO 0 STEP -1

FOR...NEXT LOOP AND STEP

By retyping (e) in program LOOP-1 and LISTing we get:

```
LIST

LOOP-1  08/12/69  11:21

10 FOR J=100 TO 0 STEP -1
20 PRINT J,
30 NEXT J
40 END
READY                (Line 20 is retyped to closely pack the numbers)

20 PRINT J;
RUN

LOOP-1  08/12/69  11:22

 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  8
 82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65
 64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47
 46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29
 28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11
 10  9  8  7  6  5  4  3  2  1  0

TIME:  0.259 SEC.
READY
```

The last example uses STEP -1, since the computer counts in a positive direction on the number line....



and we wanted to count backwards, we instructed the program to do this with STEP -1.

FOR...NEXT LOOP AND STEP (Continued)

We could write a number of things such as times tables and, odd and even numbers using STEP.

Below is a program that writes the 5 times table:

```
NEW FIVEX
READY

10 FOR X=5 TO 100 STEP 5
20 PRINT X;
30 NEXT X
40 END
RUN

FIVEX  08/12/69  11:24

 5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
 95  100

TIME:  0.085 SEC.
READY
```

Retyping line 20 with a comma (,) after X spreads the numbers into five columns.

```
20 PRINT X,
RUN

FIVEX  08/12/69  11:24

 5              10              15              20              2
 30             35              40              45              5
 55             60              65              70              7
 80             85              90              95              1

TIME:  0.141 SEC.
READY
```

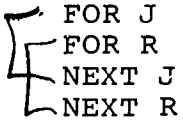
Write a program that prints the (a) 3 times table, (b) odd numbers, and (c) even numbers.

STEPs may be variables, decimals, fractions, integers, complicated formulas and negative or positive numbers.

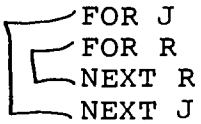
MORE ON FOR...NEXT

Of course, we could have more than one FOR...NEXT loop in a program, but they cannot cross each other.

INCORRECT

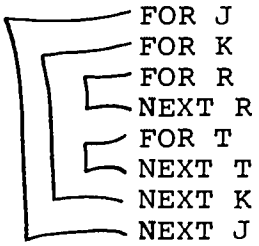


CORRECT



We could have many FOR...NEXT loops, these being nested inside each other:

CORRECT



PREPARING A TABLE OF VALUES

Often it is necessary to prepare a table of values, so that you can see the relationship between variables in an equation. In the case below, the table of values can be used to graph the line y=3X+5.

```
NEW TABLE
READY

10 PRINT "TABLE OF VALUES FOR ";
20 PRINT "Y=3X+5"
30 PRINT
40 PRINT "X","Y"
50 PRINT
60 FOR X=-10 TO 10
70 LET Y=(3*X)+5
80 PRINT X,Y
90 NEXT X
100 END
RUN

TABLE  08/12/69  11:30

TABLE OF VALUES FOR Y=3X+5

X              Y

-10            -25
-9             -22
-8             -19
-7             -16
-6             -13
-5             -10
-4             -7
-3             -4
-2             -1
-1              2
 0              5
 1              8
 2              11
 3              14
 4              17
 5              20
 6              23
 7              26
 8              29
 9              32
 10             35

TIME:  0.243 SEC.
READY
```

Write a program to print a table of values from -5 to 5 of Y=4X-2.

Write a program to print a table of values from -5 to 5 of $Y=\frac{2X}{3}+4$.

HINT: LET Y=(((2*X)/3)+4)

THE LINE and [illegible]

Each line on the teletype has exactly 75 spaces numbered from 0 - 74. By using the TABulator, we can cause information to be printed at any specific location on a line.

Suppose we want the number seven (7) to be printed in the 13th space on the line. A program that will do this si shown below:

```
NEW TAB
READY

10 PRINT TAB(12);"7"
20 END

TAB 08/12/69  11:32

              7

TIME:   0.044 SEC.
READY
```

This program brings the line pointer to space 13 (remember 0 is the first space, and 1 the second, etc.) and prints out 7. Since we used no plus (+) or minus (-) sign or space was taken before the number.

```
NEW TAB1
READY

10 PRINT TAB(12);7
20 END
RUN

TAB1  08/12/69  11:33

              7

TIME:  0.045 SEC.
READY
```

This program will print the 7 in space 14 since space 13 has an invisible (+) sign. The quotes were removed from around 7.

You, of course, could have many TABs on a line, each separated by a semi-colon (;).

THE LINE AND TAB (Continued)

Instead of TAB, we might find it useful to use a comma (,) after a PRINT. The comma brings the line pointer to the next 15th space from where it is. Commas cause the line pointer to skip across from 0 - 14, 15 - 29, 30 to 44, 45 to 59, and 60 to 74. The pointer points to either 0, 15, 30, 45, or 60 and returns to 0 on the next line.

THE LINE and COMMA

The program shown below causes the number "7" to print at the space numbered 44. (This, of course, is the 45th space on the line 0 - 74.)

```
NEW COMMA
READY

10 PRINT ,,,"7"
20 END
RUN

COMMA  08/12/69  11:33

                                                          7

TIME:  0.056 SEC.
READY
```

CHAPTER FOUR

You're really getting up there!!

Now that you have become a Level Three BASIC Programmer, you probably want to know more about BASIC.

This chapter deals with the mathematical concepts of inequalities and conditional statements called IF...THEN and ON... GO TO and GOSUB...RETURN.

In addition, a method for putting in large amounts of information either numbers or strings, is explained by the use of READ...DATA.

INEQUALITIES

In the BASIC language, as in mathematics, we have a great deal of use for inequalities. Shown below are the inequalities used in BASIC:

A<B	means A is less than B
A>B	means A is more than B
A=B	means A has the value of B
A<>B or A>><B	means A does not have the value of B
A<=B or A=<B	means A is less than or has the value of B
A>=B or A=>B	means A is more than or has the value of B

IF...THEN

The IF...THEN statement is a method for making a decision or branching. In English, it is called a conditional statement "if something happens, then something will be sure to follow."

Some examples using IF...THEN statements are shown below: (The number after THEN refers to a line number in the program)

```
NEW RAIN
READY

10 PRINT "IS IT RAINING OUTSIDE";
20 INPUT A$
30 IF A$="YES" THEN 100
40 IF A$="NO" THEN 200
50 GO TO 10
100 PRINT "READ A BOOK OR PLAY CHECKERS."
110 STOP
200 PRINT "PLAY BALL!!!   GOOD WEATHER."
999 END
RUN
```

The program asks the question IS IT RAINING OUTSIDE?

If you answer YES, then you are sent to line 100 which gives indoor activities to do. If you answer NO, then you are told to play ball because of good weather. If you type in some other word like 'DON'T KNOW' neither the IF on line 30 or line 40 is done (executed) and you drop through to line 50 which sends you back to line 10 and asks the question all over again. The stop is placed on line 110 so that you won't print the statement on line 200. There is no stop on any line after 200 since the next line is END.

```
RAIN  08/12/69  11:38

IS IT RAINING OUTSIDE? YES
READ A BOOK OR PLAY CHECKERS.

TIME:  0.076 SEC.
READY

RUN

RAIN  08/12/69  11:38

IS IT RAINING OUTSIDE? NO
PLAY BALL!!!   GOOD WEATHER.

TIME:  0.078 SEC.
READY
```

IF...THEN (Continued)

Another sample is given below:

```
NEW TEMP
READY

10 PRINT "WHAT IS THE TEMPERATURE OUTSIDE";
20 INPUT A
30 IF A<32 THEN 300
40 IF A=32 THEN 200
50 IF A>32 THEN 100
100 PRINT "THE TEMPERATURE IS NOT YET FREEZING, GET YOUR ANTIFREEZE NOW."
101 STOP
200 PRINT "IT'S FREEZING NOW, ALMOST TOO LATE FOR ANTIFREEZE."
201 STOP
300 PRINT "IF YOU DIDN'T GET ANTIFREEZE YOUR CAR IS ZAPPED!!!!"
999 END
RUN
```

This program shows the use of inequalities on line 30, 40 and 50. We really didn't need line 50 since no other case could exist if line 30 or line 40 weren't done (executed). The THEN statements were so constructed that if line 50 wasn't there, the program would still work for temperature above 32 since it would print the correct statement when it reached line 100, and then stop.

```
TEMP  08/12/69  11:42

WHAT IS THE TEMPERATURE OUTSIDE? 25
IF YOU DIDN'T GET ANTIFREEZE YOUR CAR IS ZAPPED!!!!

TIME:  0.092 SEC.
READY

RUN

TEMP  08/12/69  11:42

WHAT IS THE TEMPERATURE OUTSIDE? 32
IT'S FREEZING NOW, ALMOST TOO LATE FOR ANTIFREEZE.

TIME:  0.094 SEC.
READY

RUN

TEMP  08/12/69  11:43

WHAT IS THE TEMPERATURE OUTSIDE? 101
THE TEMPERATURE IS NOT YET FREEZING, GET YOUR ANTIFREEZE NOW.

TIME:  0.098 SEC.
READY
```

ON...GO TO

Is similiar to the IF...THEN statement, but allows a many branched switch.

ON X GOTO 100,200,300,350 means:

If X = 1 GOTO 100

If X = 2 GOTO 200

If X = 3 GOTO 300

If X = 4 GOTO 350

The X may be a complicated formula

As you become more experienced in BASIC your programs will probably become longer and more complicated.

Often you may have to do a certain routine a number of times. Suppose you would have to compute a complicated set of instructions a number of times in the same program. Instead of retyping out the set of instructions, you could place them at a suitable location, say line 850, and refer to them whenever you needed this set of procedures. This is accomplished by using the GOSUB and RETURN.

15 GOSUB 850 sends your program to line 850 and does the routine,in this case 2 lines,and then returns your program one line later when the word RETURN at the end of the routine is reached.

```
14 FOR X=1 to 10
15 GOSUB 850
16 NEXT X

    .
    .
    .
    .
840 STOP
850 LET A = (3.14) * (R↑2)
860 PRINT A, R, R 2
870 RETURN
```

Often you prepare a program, but do not know the exact information you are working with. This information is called DATA. It could be sales of teckets to the school basketball game, or election results or test grades. Anything that can be represented as either a number or word can be considered DATA.

DATA can be placed anywhere in your program, but is usually at the beginning or the end. It is suggested that DATA be placed, depending upon the size of your program, at lines 80 or 800 or 8000, etc.

READ statements use letters (variables) and can also be placed anywhere in your program. If you do not have enough DATA, or if you want the program to continue until all of the DATA is used up, the computer will print OUT OF DATA ON LINE _____, and end the program. This is a signal to you.

A program using READ and DATA is given below:

```
NEW LUNCHRM
READY

10 PRINT "SALES OF MILK AND SODA"
20 PRINT,"SEPT. 1969"
30 PRINT
40 READ A$,A
50 PRINT A$,A
60 GOTO 40
80 DATA MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
90 DATA 32.40,23,5,76.45,27.40
99 END
RUN

LUNCHRM  08/12/69  11:46

SALES OF MILK AND SODA
              SEPT. 1969

MONDAY          32.4
TUESDAY         23
WEDNESDAY       5
THURSDAY        76.45
FRIDAY          27.4
OUT OF DATA IN 40

TIME:  0.122 SEC.
READY
```

If the program has the word RESTORE, all the DATA can again be used from the beginning. RESTORE$ only restores the string DATA and RESTORE* only restores the numerical DATA.

In the program LUNCHRM, the RESTORE statement was not used.

If the words or strings are too complicated, it is sometimes necessary to put a pair of quote symbols around each particular piece of data. Examples of complicated strings are shown below:

```
800 DATA "SMITH, MR. JOHN", "SMITH, MRS. MARY"
810 DATA "SMITH, DR. & MRS. THOMAS", "336 PLEASANT AVE."
```

CHAPTER FIVE

Congratulations again!!!

We are certainly pleased that you have come this far in BASIC.

This chapter will discuss in greater detail exponents that are fractions (or decimals), something called functions and procedure of making your own functions.

Some special commands not given before will be described, plus the REMark statement. A whole group of things called the EDIT package will be explained, and of course, more math.

J = J+1,etc.

In arithmetic or algebra the statement J equals J + 1 is impossible! But in BASIC the equals sign does not have exactly the same meaning. Equals in BASIC means whatever is on the left side of the equals sign takes on the value of the right side.

A program using the idea of J = J + 1 is shown below:

```
NEW COUNTING
READY

10 LET J=J+1
20 PRINT J;
30 IF J= 100 THEN 50
40 GO TO 10
50 END
RUN

COUNTING  08/12/69  11:47

 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
 39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
 57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
 75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92
 93  94  95  96  97  98  99  100

TIME:  0.261 SEC.
READY
```

Line 10 says LET J be replaced by J+1. Since the first time through the J in J+1 was not assigned a value the computer said to itself that it was zero. Therefore line 10 LET J=J+1 gave the J on the left of the equals sign the value of 0+1.

Line 20 prints J or in this case the number 1, close packed because of the semi-colon(;).

Line 30 is checking statement if J is 100 then the program would go to line 50 or END. The J was not 100.

Line 40 sends the program to line 10 with J now equal to 1 not zero as before.

Line 10 says J=J+1 but now the J on the right is a 1, so 1+1 is 2. J on the left now has the value of 2.

Line 20 prints J...in this case the number 2.

Line 30 checks if J is 100! J is 2 not 100.

J=J+1 (Continued)

Line 40 sends program back to 10.

Line 10 now says Let J=J+1, but the J on the right side is a 2. Therefore 2+1 is 3.

Line 20 prints J...in this case the number 3.

Line 30 checks if J is 100. J is 3 not 100.

Line 40 sends program back to 10.

This continues until J = 100 and the check on line 30 sends the computer to line 50 or END.

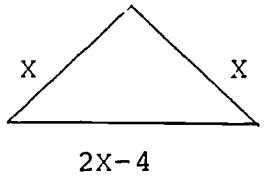
MORE ON J=J+1

An excellent use for the idea of (concept) J=J+1 is problem solving. Below are a number of problems and solutions using the idea of J=J+1 in BASIC.

PROBLEM 1

The base of an isosceles triangle is 4 feet less than the sum of the two equal sides. The perimeter of the triangle is 76 feet. Find the length of each side. (X is a whole number.)

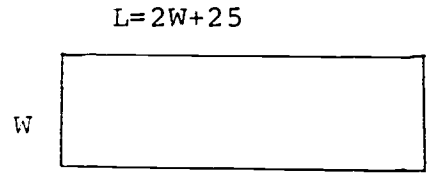
```
NEW
NEW FILE NAME--NEW PBLM1
READY

10 LET X=X+1
20 LET P=X+X+2*X-4
30 IF P=76 THEN 50
40 GO TO 10
50 PRINT X,X,2*X-4
60 END
RUN

NEW  08/12/69  11:49

 20            20            36

TIME:  0.064 SEC.
READY
```



PROBLEM 2

The length of a yard exceeds twice its width by 25 feet, and 950 feet of fencing are needed to enclose it. Find its dimensions. (W is a whole number.)

```
NEW PBLM2
READY

10 LET W=W+1
20 LET L=2*W+25
30 LET P=2*L+2*W
40 IF P=950 THEN 60
50 GO TO 10
60 PRINT W,L,P
70 END
RUN

PBLM2  08/12/69  11:52

 150           325           950

TIME:  0.074 SEC.
READY
```



MORE ON J=J+1 (Continued)

PROBLEM 3

John and Otto picked 33 quarts of cherries. The number picked by John exceeded half the number picked by Otto by 3. How many did each pick?

```
NEW PBLM3
READY

10 LET O=O+1
20 LET J=(.5*O)+3
30 LET N=J+O
40 IF N=33 THEN 60
50 GO TO 10
60 PRINT "JOHN= "J,"OTTO= "O, "TOGETHER= "N
70 END
RUN

PBLM3  08/12/69  11:54

JOHN= 13                 OTTO= 20                 TOGETHER= 33

TIME: 0.071 SEC.
READY
```

An exit check should be inserted in programs PBLM1, PBLM2 and PBLM3 in the event that the answer is not a whole number, a sample exit check for PBLM3 is shown below:

```
45 IF N>33 THEN 65
64 STOP
65 PRINT "N>33, ANSWER NOT A WHOLE NUMBER"
```

Write an exit check for PBLM1 and PBLM2.

RADIAN MEASURE, ANGLES and π(Pi)

Of course you are aware that you could measure angles with something called a protractor.

The BASIC language does not use angles, but does use something called a radian. Before we discuss this function further you have been introduced to the symbol Pi (π) back in 6th, 7th, and 8th grades.

Do you remember what π stood for?

Sure you do: π was almost equal to 3 1/7 or 22/7 or 3.14.

The teletype does not have the symbol π on the keyboard so we are going to have to use a decimal. We use the following decimal for greater accuracy: 3.14159265

π = 3.14159265...for some purposes we need π to more decimal places

The area of a circle is π times the radius times the radius or:

$A = \pi R^2$

In BASIC this would be:

```
LET A = (3.14159265)*R↑2
```

The circumference of a circle was either:

$C = \pi D$ or $C = 2\pi R$

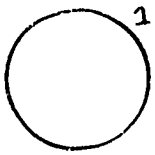
In BASIC you would write:

```
LET C = (3.14159265)*D
```

or

```
LET C = 2*(3.14159265)*R
```

A circle, of course has 360°, so if we start at point 1 and go all around the circle and return to point 1, we have then made a complete tour around the circle.



We could say we went 360° or better still 2π. Since 2π would seem to mean 360° or the circumference of the circle.

From this we could easily form the following table:

$2\pi = 360°$

$\pi = 180°$

$\pi/2 = 90°$

$\pi/4 = 45°$

$\pi/6 = 30°$

$\pi/18 = 10°$

I'm certain you could figure π/3 is 60° or 4π is 720°.

We now have a new meaning for π.

$\pi = 180° = 3.14159265$ (radian)

By careful calculation we can find that a specific angle smaller than 60°has the value 1. This is called a radian.

One radian $> 57°$

$< 58°$

$57° < (\text{one radian}) < 58°$

All of the angles in the BASIC language are written in radian measure. Therefore it is necessary to convert radians to angles and angles to radians.

We use the following proportion to do these calculations:

$$\frac{180°}{\text{angle}} = \frac{3.14159265}{X}$$

If we have an angle of 90° and want to find its value in radians, we would do the following:

$$\frac{180°}{90°} = \frac{3.14159265}{X}$$

or

$$180° * X = 3.14159265 * 90°$$

or

$$X = \frac{3.14159265 \times 90°}{180°}$$

A BASIC program that does this (FOR 57.296°) is:

```
NEW X
READY

10 PRINT "WHAT ANGLE DO YOU WISH CONVERTED (LEAVE OFF DEGREE MARKS)"
20 INPUT A
30 LET X=(3.14159265)*A/180
40 PRINT X
50 END
RUN

X   08/12/69  17:05

WHAT ANGLE DO YOU WISH CONVERTED (LEAVE OFF DEGREE MARKS)
? 57.296
 1.

TIME:  0.079 SEC.
READY
```

We could make the answer a little fancier by replacing line 40 with:

```
40 PRINT A"  DEGREES = "X" RADIANS."
RUN

X  08/12/69  17:09

WHAT ANGLE DO YOU WISH CONVERTED (LEAVE OFF DEGREE MARKS)
? 57.296
 57.296 degrees = 1. RADIANS

TIME: 0.092 SEC.
READY
```

SINE, COSINE and TANGENT

In 9th year math or algebra we use functions called Sine, Cosine and Tangent. We must know how to convert angles to radians to use the computer in these problems.

SIN (X)

COS (X) — All refer to X as radian measure.

TAN (X)

The sine of 45° is written:

Sine $(\frac{\pi}{4})$ or Sine $(\frac{3.14159265}{4})$ or Sine (.785381625)

You can construct your own Sine, Cosine and Tangent tables using the following program:

```
NEW SOHCAHTOA
READY

SOHCAHTO  08/12/69  17:13

5 PRINT "ANGLE","RADIAN,"SINE,"COSINE","TANGENT"
10 FOR A=0 TO 90 STEP 10
20 LET X=(3.14159265)*A/180
30 PRINT A,X,SIN(X),COS(X),TAN(X)
40 NEXT A
99 END
READY

RUN

SOHCAHTO  08/12/69  17:13

ANGLE       RADIAN       SINE        COSINE        TANGENT
 0           0            0           1             0
 10          0.174533     0.173648    0.984808      0.176327
 20          0.349066     0.34202     0.939693      0.36397
 30          0.523599     0.5         0.866025      0.57735
 40          0.698132     0.642788    0.766044      0.8391
 50          0.872665     0.766044    0.642788      1.19175
 60          1.0472       0.866025    0.5           1.73205
 70          1.22173      0.939693    0.34202       2.74748
 80          1.39626      0.984808    0.173648      5.67128
 90          1.5708       1.          1.58933 E-8   6.29198 E+7

TIME:  0.360 SEC.
READY
```

REMARK also (APOSTROPHE ')

The REMark or REM statement has no effect upon the running of the program. It appears only on a listing as an aid to the programmer.

On long or complicated programs, the REM statement is used to explain to the programmer what is happening.

An apostrophe (') does the same thing as a REM statement and is quicker to type.

An incomplete program using the REM and (') is shown below:

```
10 LET X=(A*3.14)/180   ' CONVERTS DEGREES TO RADIANS
20 REM                    PRINTS OUT A TABLE OF SINES
30 FOR A =1 to 90       ' START OF LOOP IN A
40 PRINT A, SIN(X)      ' OUTPUT A, SIN(X)
   .
   .
   .
   ETC.
```

(Special Note: REM's can only be used immediately after a line number. In all other places, use the apostrophe (').)

FUNCTIONS

The BASIC language provides many other useful tools called functions. As you learn more Mathematics it will become easier for you to understand the use of these functions.

(The definitions below are not strictly defined nor listed completely-see full edition of BASIC Manual for a complete description.)

FUNCTIONS	INTERPRETATION	
SIN(X)	Find the sine of X	X interpreted as a number, or as an angle measured in radians
COS(X)	Find the cosine of X	
TAN(X)	Find the tangent of X	
COT(X)	Find the cotangent of X	
ATN(X)	Find the arctangent of X	
EXP(X)	Find e^x	
LOG(X)	Find the natural logarithm of X (ln x)	
ABS(X)	Find the absolute value of x; $\|x\|$	
SQR(X)	Find the square root of x ($\sqrt{x}$ or $x^{1/2}$)	
INT(X)	Gives the greatest integer not greater than X INT (1.38)=1 INT (12.99)=12 INT (-2.65)=-3	
RND	Produces random numbers	
SGN(X)	Produces 1,0 or -1 SGN (6.25)=1 SGN (0)=0 SGN (-3.15)=-1	
NUM	Counts the number of numbers after a MAT input (LET N = NUM)	
TIM	Gives running time of program in seconds	
CLK$	Has the value of the time (16:26:46)	
DAT$	Has the value of the date (06/23/69)	
USR$	Has the value of the user number	
VAL(S$)	Converts the string (S$) to a number	
LEN (S$)	Counts the letters in a string	
ASC (CHAR) or ASC (ASCII abbreviation)	Converts the letters in ASCII value	

An example of how to use one of these functions (SQR(X)) is shown below:

```
NEW SQROOT
READY

10 PRINT "THIS PROGRAM PREPARES A TABLE OF NUMBERS,SQUARES AND  ";
20 PRINT "SQUARE ROOTS."
30 PRINT "

          "
40 PRINT "NUMBER","SQUARE","SQUARE ROOT"
50 PRINT "------","------","-----------"
60 PRINT "
           "
70 FOR J=1 TO 25
80 PRINT J,J↑2,SQR(J)
90 NEXT J
99 END
RUN

SQROOT      08/12/69   17:18

THIS PROGRAM PREPARES A TABLE OF NUMBERS,SQUARES AND  SQUARE ROOTS.


NUMBER          SQUARE          SQUARE ROOT
------          ------          -----------


 1               1               1
 2               4               1.41421
 3               9               1.73205
 4               16              2
 5               25              2.23607
 6               36              2.44949
 7               49              2.64575
 8               64              2.82843
 9               81              3
 10              100             3.16228
 11              121             3.31662
 12              144             3.4641
 13              169             3.60555
 14              196             3.74166
 15              225             3.87298
 16              256             4
 17              289             4.12311
 18              324             4.24264
 19              361             4.3589
 20              400             4.47214
 21              441             4.58258
 22              484             4.69042
```

23	529	4.79583
24	576	4.89898
25	625	5

ROUNDING OFF (DECIMAL PLACES)

Using the INTeger function we are able to round-off numbers to the nearest tenth (one decimal place) or nearest hundredth (two decimal places) or to whatever number of decimal places wanted.

```
10 LET X= (some number)
20 PRINT INT(10*X+.5)/10↑2
30 END
```

Will round-off a number to two decimal places.

The following program will round-off a number to any number of decimal places needed:

```
10 LET X= (some number)
20 LET D= (number of decimal places)
30 PRINT INT(X*10↑D+.5)/10↑D
```

DEF

Often it is necessary for you to create your own functions. You can do this in each program using a DEFine statement.

Of course as an alternate to this procedure you could always use a GOSUB and RETURN as previously discussed.

THE DEF STATEMENT is always followed by a space and the letters FNA or FNB or FNC, etc. All the way to FNZ.

Therefore you can create up to 26 DEF FN'S in your program for numbers. The two programs below print out a table of sines.

```
NEW SINETAB1
READY

10 DEF FNA(X)=(X*3.14159265)/180
20 DEF FNB(X)=SIN(FNA(X))
30 PRINT "ANGLE","SINE"
40 FOR X=0 TO 180 STEP 10
50 PRINT X,FNB(X)
60 NEXT X
70 END
RUN

SINETAB1  08/12/69  17:23

ANGLE          SINE
 0              0
 10             0.173648
 20             0.34202
 30             0.5
 40             0.642788
 50             0.766044
 60             0.866025
 70             0.939693
 80             0.984808
 90             1.
 100            0.984808
 110            0.939693
 120            0.866025
 130            0.766044
 140            0.642788
 150            0.5
 160            0.34202
 170            0.173648
 180            3.614 E-9

TIME:  0.277 SEC.
READY
```

or

```
NEW SINETAB2
READY

10 DEF FNC(Y)=SIN (Y*3.14159265/1
30 PRINT "ANGLE","SINE"
40 FOR Z=0 TO 180 STEP 10
50 PRINT Z,FNC(Z)
60 NEXT Z
70 END
RUN

SINETAB2  08/12/69  17:26

ANGLE          SINE
 0              0
 10             0.173648
 20             0.34202
 30             0.5
 40             0.642788
 50             0.766044
 60             0.866025
 70             0.939693
 80             0.984808
 90             1.
 100            0.984808
 110            0.939693
 120            0.866025
 130            0.766044
 140            0.642788
 150            0.5
 160            0.34202
 170            0.173648
 180            3.614 E-9

TIME:  0.267 SEC.
READY
```

Please consult the regular BASIC Manual for more information about the use of DEF.

The square root of 25 is 5.
The square root of 36 is 6.
The square root of 1 is 1.

We can define the square root as "what number times itself makes" 81? the answer is 9.

The square root of 100 is 10.

The square root of 50 is ... a decimal larger than 7, but smaller than 8. Why?

No number times itself exactly equals 50. You can say 5 times 10 is 50, but this isn't a number times itself. Seven times seven is 49, eight times eight is 64. Then of course the square root of 50 must be more than seven and less than eight.

What is the square root of 50?

We can use the SQR(X) function to find square root. A program to do this is given below:

This program finds the square root of 50:

```
NEW ROOTS
READY

10 PRINT SQR(50)
20 END
RUN

ROOTS  08/12/69  17:27

 7.07107

TIME:  0.042 SEC.
READY
```

We could expand the idea of square root!!!

In fact, we could think of square root as the inverse of the exponent ^{2}. Below is a table of squares and square roots.

N	N^2	$\sqrt{N}$ or N↑(1/2)
Number	Square	Square Root
1	1	1
2	4	1.414
3	9	1.732
4	16	2
.	.	.
.	.	.
.	.	.

We can easily write a program to prepare this table:

```
NEW ROOT1
READY

10 PRINT "N","N↑2","N↑(1/2)"
20 PRINT"
         "
30 FOR N=1 TO 10
40 PRINT N,N↑2,N↑(1/2)
50 NEXT N
60 END
RUN

ROOT1      08/12/69   17:29

N                N↑2             N↑(1/2)


 1               1               1
 2               4               1.41421
 3               9               1.73205
 4               16              2.
 5               25              2.23607
 6               36              2.44949
 7               49              2.64575
 8               64              2.82843
 9               81              3.
10               100             3.16228

TIME:   0.226 SEC.
READY
```

Since we defined square root as N↑(1/2) or N raised to the 1/2 power (exponent). We can think of cube root as N↑(1/3)

ROOT	BASIC EXPRESSION	INTERPRETATION
SQUARE ROOT	N↑(1/2)	SQUARE ROOT OF N
CUBE ROOT	N↑(1/3)	CUBE ROOT OF N
FOURTH ROOT	N↑(1/4)	FOURTH ROOT OF N
FIFTH ROOT	N↑(1/5)	FIFTH ROOT OF N
.	.	.
.	.	.
.	.	.

CUBE ROOT DEFINITION

"What number times itself, three times" makes 8:

N * N * N = 8 The cube root is 2.

"What number times itself, three times" makes 27?

N * N * N = 27 The cube root is 3.

Can you write a simple definition for the 4th root or N↑(1/4)?

Listed below are some other commands that you might find useful:

'LIST (space) (line number)'

allows you to list a part of your program starting at a specific line number. (i.e. LIS 320)

'SYS'

allows you to change the system from the BASIC language. (i.e. FORTRAN, ALGOL, LAFFF)

'CAT'

allows you to get a list of all of your program names (or files) that you have saved.

'RENAME'

allows you to rename a program without destroying it.

'SCRATCH'

allows you to destroy your current program and let its name remain.

'(RETURN KEY)'

allows you to find out how long your program has been running after you type RUN.

'OLD DARTCAT***'

You can only LIST this file. LISTs all subjects filed in Dartmouth College Computer Library. From this you then LIST the subject to get specific programs.

'(CONTROL SHIFT P)'

Stops your program when nothing else works, also called break.

'TTY'

Supplies information about your teletype. (i.e. teletype number, programmer, language, status).

'CATALOG'

May also be called with special option codes these are:

LEN (length)
DLU (date last used)
DLM (date last modified)
ALL (all options)
SEL (starts a request for specific file names)

CHAPTER SIX

ADVANCED BASIC

This chapter contains information on the more advanced concepts and ideas in the BASIC language. Details on how to use these concepts are obtainable by calling the specific information from the computer. (i.e. EDIT) or by consulting the regular edition of the BASIC manual.

This chapter includes such topics as secret passwords, debugging programs, the EDIT package, MATRIX and determinants, files, random numbers and flags.

PASSWORDS

Certain user numbers have passwords. Passwords for user numbers are available from your teacher.

Programs (or files) can have passwords, too!

(Caution: Since only you know what the password is, and if you use too many passwords, and forget what password belongs to what program the programs are lost to you and everybody else.)

Therefore, please use the same password on all your programs. Most programs don't need passwords at all. To place a password on your program do the following:

```
NEW JOE
READY
SAVE JOE,PASSWORD:
```

The 'password' is assigned to the program JOE.

Your teacher has a special book in which passwords are recorded for each student. Students may get and use as many passwords as necessary.

DEBUGGING

If your program seems to have errors (or bugs), then it needs correction (debugging.)

First, retype RUN for error messages.

Second, correct error messages. Type LIST.

Third, retype RUN for additional error messages.

Fourth, correct error messages again. Type LIST.

Fifth, continue until no error messages appear on RUN.

Sixth, if program still doesn't work, recheck for missing PRINT statements and incomplete DATA.

Seventh, ask for help!

Your teacher has a special TRACE program available

The EDIT (or EDI) functions available are listed below:

EDIT	APPEND	DELETE	DESEQUENCE	LIST	GMD
EXPLAIN	INSERT	EXTRACT	RESEQUENCE	LOCATE	LIFE
HELP	JOIN	MOVE	SEQUENCE	PAGE	JGK
				STRING	SID

To find out how to use any of these functions type:

EDIT EXPLAIN (name of function)

or

EDI EXP (name of function)

For more information about EDIT in general type:

EDIT EXPLAIN EDIT

(Mathematicians and computer programmers enjoy themselves and like a little bit of entertainment. Four of the functions listed above are for this purpose. Can you find them?)

MATRIX and DETERMINANTS

A set of instructions for the use of the many MATRIX statements are available from your teacher, or from the regular BASIC Manual (latest edition.)

If you are interested in finding out about this topic, please ask your teacher.

FILES

A set of instructions for the use of the many FILE statements are available from your teacher, or from the regular BASIC Manual (latest edition.)

If you are interested in finding out more about this topic, please ask your teacher.

ERROR MESSAGES

See the regular BASIC Manual for an explanation or description of the many error messages that DTSS provides. Listings of the error messages and their explanations are posted near the teletype.

FLAGS

'Flags' are used by programmers to signal certain changes in routine. Often the numbers -1 or 0 are used as a flag.

The program below would eventually produce an OUT OF DATA message when RUN.

```
NEW FLAG
READY

10 READ A
20 PRINT A;
30 GO TO 10
80 DATA 1,3,4,5,6,7,2,5,6,7
99 END
RUN

FLAG       08/2/69    11:46

 1  3  4  5  6  7  2  5  6  7
OUT OF DATA IN 10

TIME:  0.071 SEC.
READY
```

By inserting the two lines below, the program is given a FLAG to finish the program without printing the OUT OF DATA message.

```
15 IF A=-1 THEN 99            'CHECKING FOR FLAG -1
90 DATA -1                    'LAST PIECE OF DATA IS SET TO -1
LIST


FLAG      08/20/69

10 READ A
15 IF A=-1 THEN 99            'CHECKING FOR FLAG -1
20 PRINT A;
30 GO TO 10
80 DATA 1,3,4,5,6,7,2,5,6,7
90 DATA -1                    'LAST PIECE OF DATA IS SET TO -1
99 END
READY

RUN

FLAG       08/20/69  11:48

 1  3  4  5  6  7  2  5  6  7

TIME:  0.071 SEC.
READY
```

RND and RANDOMIZE

RND produces a sequence of numbers between .000000 and .999999 from a table of random numbers. Each time you run RND the same sequence of numbers will be prepared.

Shown below is a program using RND:

```
NEW RND
READY

10 PRINT RND,
15 GO TO 10
99 END
RUN

RND        08/20/69   09:53

 0.406533        0.927599        0.264283        0.789368        0.976272
 0.948228        0.165784        0.328597        0.552183        0.615669
 0.912571        0.512762        0.53556         0.825354        0.777282
 0.907836        0.884522        9.99165 E-2     0.883958        0.109132
 0.742572        0.362751        0.216531        0.858972        0.133681
0.420067         0.786135

STOP

TIME:  0.999 SEC.
READY
```

RANDOMIZE when used with RND produces a random sequence of numbers between .000000 and .999999. A different sequence of numbers will be prepared each time.

Shown below is a program using RANDOMIZE with RND:

```
NEW RANDOMIZ
READY

5 RANDOMIZE
10 PRINT RND,
15 GO TO 10
99 END
RUN

RANDOMIZ   08/20/69    09:54

 0.867272        0.131017        0.246894        0.578099        0.527731
 0.659141        0.905874        0.781341        0.856027        0.182354
 0.14529         0.126799        0.428908        0.584435        0.935397
 0.43124         0.330366        0.425612        0.618403        0.217188
 0.929499        0.296386        0.624037        0.735241

STOP
```

By using the round-off procedure and INT function a sequence of numbers to the nearest tenth, hundredth, or whole number could be produced.

The following program will produce a sequence of numbers from the random number table between 0 and 99.

```
NEW RNDINT
READY

10 PRINT INT (RND*100);
20 GO TO 10
99 END
RUN

RNDINT    08/20/69  11:33

 40  92  26  78  97  94  16  32  55  61  91  51  53  82  77  90  88  9
 88  10  74  36  21  85  13  42  78  31
```

Can you write a program to produce a sequence of numbers between 0 and 999?

Can you write a program to produce a different sequence of numbers between 0 and 999 each time?

APPENDIX A

SOME SUGGESTIONS FOR STUDENT PROGRAMS

The following is taken from a list of computer programming ideas in varying degrees of difficulty prepared by Jean H. Danver under the NSF-Dartmouth Secondary School Project (NSF Grant GW-2246). For a full copy of this TOPIC OUTLINE, please contact:

KIEWIT COMPUTATION CENTER
Dartmouth College
Hanover, New Hampshire 03755

1. Write a program that will print out your name.

2. Write a program to find the product of two numbers.

3. Write a program that will read successive pairs of numbers and, on each pass, will print the numbers and their sum.

4. Write a program to read and compute the sum of the first 12 even integers.

5. Write a program to compare two numbers. If the first is larger than the second print, "NOT LESS THAN OR EQUAL". Otherwise print, "LESS THAN OR EQUAL TO".

6. Write a program to generate and compute the first ten integers and their cubes.

7. Write a program to find the sum of pairs of numbers. Print out each number and the sum in appropriate headed columns.

8. Write a program to divide any two numbers.

9. Read a list of numbers and print them out in as few rows as possible.

10. Read a list of numbers and print out every other number.

11. Read a list of numbers and print them out in two columns:

    a) As close together as possible
    b) As far as possible
    c) Somewhere in between

12. Write a program that will generate the first ten integers, compute their squares and print out in columns headed:

    "[illegible]", "[illegible]", "Sum of Squares".

13. [illegible] to generate the first 10 integers, compute

their square roots, print out the number and its square root in appropriately labeled columns.

14. Print out the numbers 1-30 in a) 5 columns, b) 7 columns, c) like this: 12345 678910

1112131415 1617181920

2122232425 2627282930

15. Add up the squares of odd numbers for 101 to 201.

16. Write a program to find the sum and products of pairs of numbers. Print out each number, the smaller first in appropriate headed columns. Arrange the printout so that the results of the last pair are printed first and the first pair printed last.

17. Consider the numbers .5 thru 5 in steps of .5 inclusive. Write a program that will center a three column table on the paper where the first column contains the numbers, the second column contains the fifth powers of the numbers, and the third column contains the fifth roots. Also, have headings for each column.

18. Write a program that will have the computer center a three column table on the page. The three columns should have headings and should contain the entries X, X to the fourth power, and the fourth root of X, where X takes on the values .5, 1.5, 2, .. 4.5, and 5.

19. Write a program to compute absolute value without using the command.

20. Write a program to round off numbers to the nearest 10, 100, 1000, 1/10, 1/100.

21. Write a program to round off numbers to any place desired.

22. Read any three numbers and print them out in descending (ascending) order.

23. Determine if one number is divisible by another.

24. Print out all integers between 1 and 100 which are:

    1. Divisible by 3 and 5
    2. " " 13
    3. " " 31

Also, find the sums of the numbers in each group.

SOME SUGGESTIONS FOR STUDENT PROGRAMS (Continued)

25. Compose a program which will find the largest factor of any number.

26. Determine the common factors of any two given numbers.

27. Write a program to determine the greatest common divisor (GCD) and lowest common multiple (LCM) of any two numbers.

28. Factor integer using the Method of Fermat.

29. Compute the greatest common divisor (GCD) of 2 given numbers through the use of the Euclidean Algorithm.

30. List the prime numbers up to a given number, N.

31. List the prime numbers between any two given numbers, N and M.

32. Express any number as a product of its prime factors.

33. Find the prime factors of a given number.

34. List all primes which are the sum of squares.

35. Program the Sieve of Eratosthenes.

36. Find all the pairs of twin primes between any two numbers.

37. Test numbers for primeness by the use of Wilson's Theorem ...n is prime if the only if (n-1) ! = -1 mod(n).

38. List N, N! and 1/N in 3 columns.

39. Find the sum of the first N odd numbered even numbers.

40. Locate the largest number in a sequence of numbers and its position in the sequence.

41. Find the smallest, the largest, and the difference between the smallest and the largest of a list of numbers.

42. Order a list of numbers.

43. Construct a table of squares and cubes of the multiples of 3 from 12 to 42.

44. Write a program to compute N↑E where N = any number and E = any integer without using the operator↑.

45. Write a program to compute $(X*Y)^2$ without using the command↑ or *.

46. Write a program using the random number generator to

generate 25 random integers between 1 and 100. Then print out the list of these integers from the smallest to the largest.

47. Change fractions to decimals.

48. Write a program which will convert linear measures in the metric system (meters and centimeters only) to equivalent measures in the English system (feet and inches only). INT(X) may be useful here.

49. Write a program to play the following game: The computer tries to guess a number you have in mind from one to 100. First, it guesses a number and you tell it if the number is too high or too low or correct. On the basis of the information you give, the computer guesses again. This continues until the computer guesses right!

50. Compose a program which will supply the decimal equivalents to the rational numbers 1/11, 2/11,..., 10/11. On the first pass through the program the equivalents should be rounded off to the nearest hundredth, on the second pass to the nearest thousandth, and on the third pass to the nearest ten-thousandth.

51. Program a general conversion between arbitrary bases.

52. List Pythagorean triplets.

53. List numbers which are the sums of two squares up to any given number.

54. Write a program to compute N mod M.

55. Write a program to determine if two numbers are congruent in mod M.

56. Print out modular arithmetic tables.

57. Program an algorithm to convert numbers from decimal to octal to binary.

58. Write a program to determine the solutions of a quadratic equation.

59. Write a program to solve 1st degree equations in one unknown.

60. Determine the slope of a line given any two points.

61. Find the square root of a number without using the operator SQR.

SOME SUGGESTIONS FOR STUDENT PROGRAMS (Continued)

62. Program Newton's Method for approximating square roots. This is a guess and then averaging the divisor and quotient for a new guess.

63. Modify Newton's Method and approximate cube roots.

64. Write a program to calculate the sum of the first N terms of a geometric progression.

65. Write a program to generate a list of numbers which is the sum of corresponding elements of two other lists of equal number of elements.

66. Find the perimeter and area of various geometric figures.

67. Find the volume of various geometric figures.

68. Write a program to solve percentage word problems.

69. Change integers to Roman Numerals.

70. Read a four digit number. Print out the number and the number of times the digit 7 appears in the number.

71. Find sets of 5 numbers greater than zero which have a sum of 1000.

72. Given the coordinates of 4 points, determine whether they form the vertices of a square, a rhombus, a rectangle or a quad.

73. Given two sets A,B, compute AUB and AΩB.

74. Write a program which tells if two sets are equal.

75. Write a program to solve the triangle problem--i.e., to find the sum of the perimeters of triangles inscribed in a 4 inch equilateral triangle if you continuously counted the mid-points of previous triangles.

INDEX